(4)

AR-006-401

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Communications Division

REPORT
ERL-0511-RE

CONTROL OF A COSSOR CSP1250 IONOSPHERIC SIMULATOR IN AN AUTOMATED HF TEST BED

by

Patricia L. Slingsby

SUMMARY

This document describes software developed to facilitate PC control of the Cossor CSP1250 Ionospheric Simulator incorporated into the Electronics Research Laboratory Automated HF Test Bed.

MAY 1990

COPY NO. 15

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1600, Salisbury, South Australia, 5108.

CONTENTS

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

CCIR	International Radio Consultative Committee
DDS	Development Documentation System
DSTO	Defence Science and Technology Organisation
ERL	Electronics Research Laboratory
PC	Personal Computer
PTS	Pilot Tone Sounder

1 INTRODUCTION

The development of modems, scramblers, frequency hopping units etc, for HF channel operation requires repeatable and comparative tests to be carried out under typical conditions. By using HF channel ionospheric simulators these tests can be carried out more cost-effectively and conveniently without the need for the relatively expensive live-link tests which would otherwise be necessary. Such simulators not only provide repeatable HF channel conditions, they also permit some control of the link parameters - an aspect not readily achievable with "live" circuits.

The Automated HF Test Bed currently under development at the Electronics Research Laboratory (ERL) uses a Cossor CSP1250 Ionospheric Simulator. The Test Bed also incorporates transmit and receive modems, data error analysers (which generate and monitor a pseudo-random bit sequence), and a Pilot Tone Sounder (which performs bit-error-rate analysis of the received data). A functional diagram of the ERL Automated Test Bed is provided in Figure 1 overleaf.

NOTE: *It is intended in the future to modify the test bed so that the data collection and analysis functions of the Pilot Tone Sounder will be incorporated into the simulator control software. Figure 2, overleaf, shows the proposed modified configuration of the test bed.*

The Test Bed is to be used primarily as a research tool and a calibration facility for an Intelligent Frequency Management System. It will also be used in a terrestrial transmission link model to be developed under the proposed Innovative Communication Architectures task. In addition, the Automated HF Test Bed will provide a useful tool for the development and testing of HF communications equipment.

The propagation model upon which the Cossor simulator is based is that defined by the International Radio Consultative Committee (CCIR) (HF Simulator model in question 21/3 of the Geneva symposium). The simulator can reproduce various propagation effects, including differential multipath delay, frequency offset, fading and noise, over either one or two simulated HF links.

Normally the Cossor CSP1250 simulator is controlled by an Epson HX20 microcomputer using a program written in Epson Basic. For its use in the ERL Test Bed the simulator is to be controlled by an IBM compatible PC via an RS232C port running at 4800 baud.

This document describes the software developed to support this application.

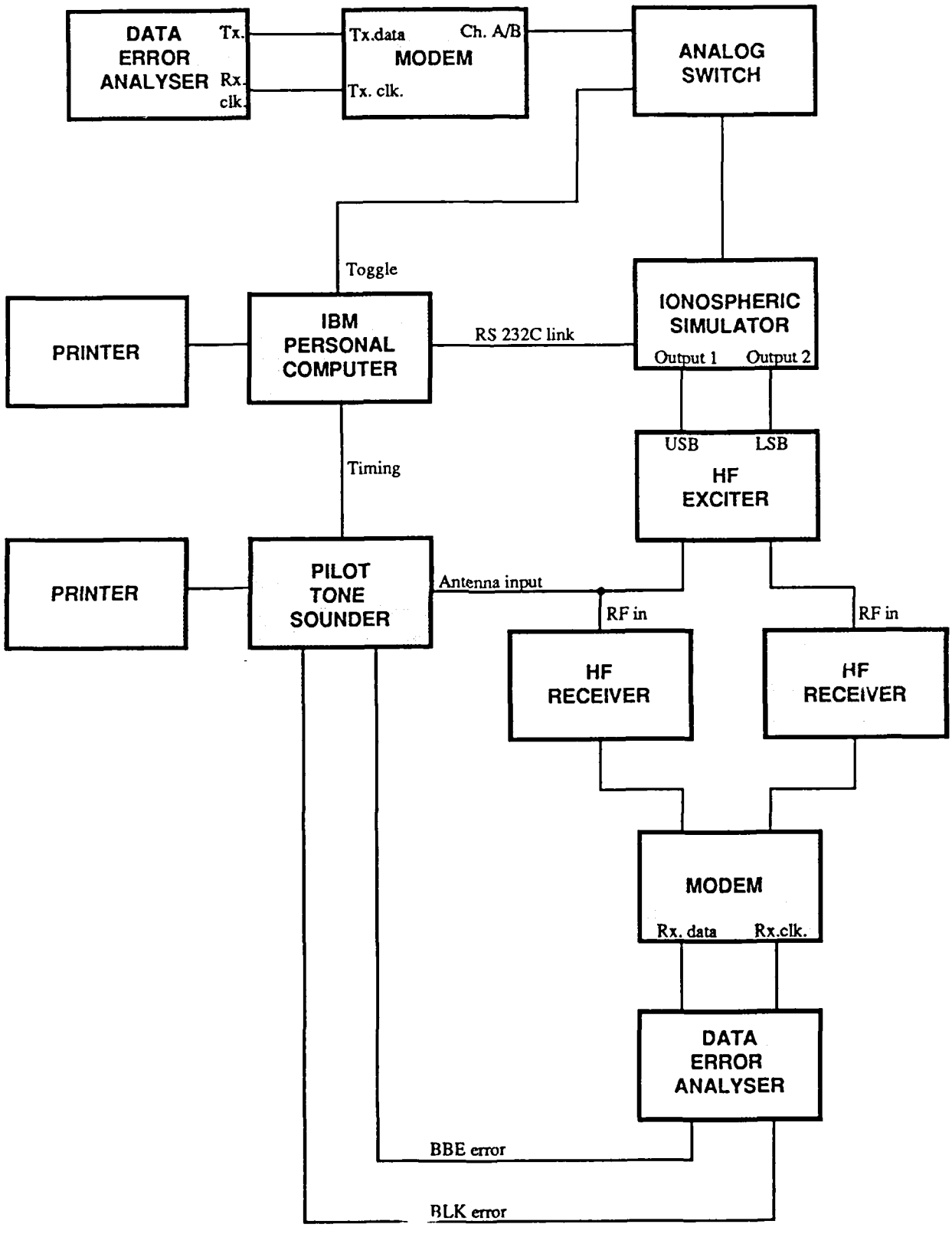

Figure 1 Configuration of the ERL Automated HF TestBed

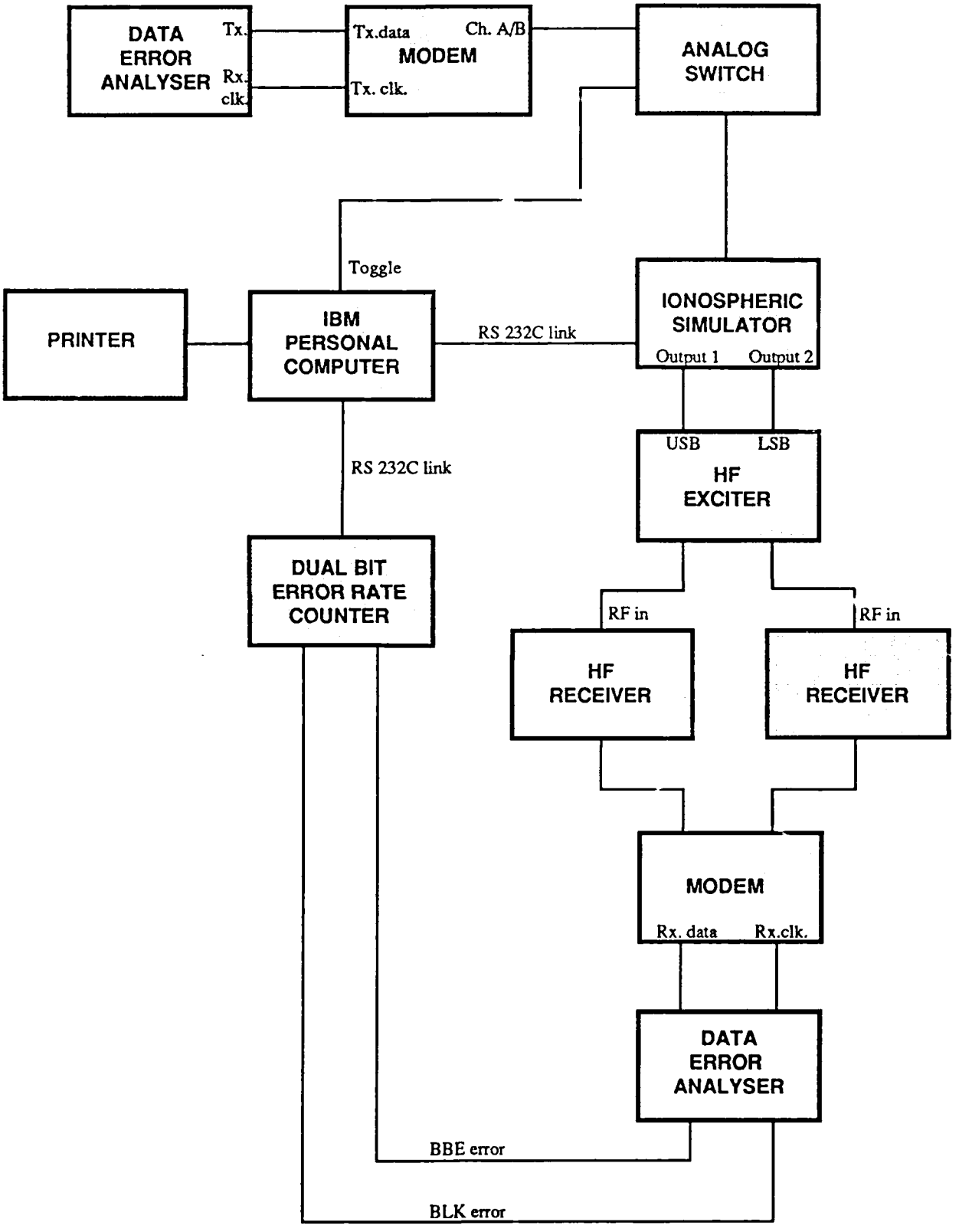

Figure 2 Proposed modified configuration for the ERL Automated HF Test Bed

2 ADVANTAGES OF THE PC CONTROL PROGRAM

The ERL-developed control program for the simulator is modelled on the program supplied by Cossor in support of the Epson micro computer with which the unit is normally controlled. In addition to retaining the elementary features and capabilities of the Epson package the ERL PC program provides a much enhanced more user-friendly operating environment for the test bed operator. Also, many of the limitations of the Basic code are alleviated by the test bed PC package which additionally facilitates the automated running of test configurations that have been stored on disk. The PC package offers an improved display and access to more memory than does the HX20 machine, with additional file and data manipulation capabilities. At some later stage it will also include a data collection and analysis facility.

The PC program enables the operator to edit, display and otherwise manipulate all of the path characteristics of the simulator. It is menu driven and includes comprehensive help menus and data manipulation utilities. Unlike the Epson program for which the operator is required to be familiar with a number of specific keyboard responses and codes for the numerous controller options, this package displays menu options at each sub-menu level. Menu options are activated by function keys rather than by the specific ASCII codes of the Epson software, and the operator is given flexibility to move between stages of the executing program.

The PC package is coded in Turbo C (version 1.5) with the asynchronous controller-simulator I/O modules written totally in assembly code and compiled separately. I/O over the RS232C link is interrupt-driven with transmit and receive data being stored in circular buffers which may be rewound and accessed as required.

3 THE SIMULATOR MODE OF OPERATION

The Cossor simulator can be set up in either of the following two test configurations.

(a) One groundwave path plus four skywave paths, each of which can have a special path delay and Rayleigh fade associated with it.

(b) Two groundwave paths which can be frequency offset from each other, plus two skywave paths associated with each groundwave. These paths can have various fade, noise and delay parameters assigned to them. In addition, a single oscillator tone (the frequency of which can be specified by the operator) can be added to the signal and the gains of each path can be changed.

The simulator also has an additional input to which an interference source or a recorded signal may be added. This facility is useful when evaluating the performance of systems in the presence of noise.

4 THE SIMULATOR CONTROL PACKAGE

The simulator control package gathers from the operator all the parameter data required to be set up on the simulator for a test (or set of tests). This data includes relative strengths of groundwave and skywave signals, the signal-to-noise ratios for each path, frequency offset between groundwave paths, skywave fade rates and skywave delay times. The operator also specifies the time for each increment of the test parameter(s) and the value by which that parameter(s) should be incremented or decremented.

The control program scales this user data and calibrates it to conform with the simulator data requirements. As part of this calibration the simulator measures the input signals that are applied to it so that it can use these values to adjust internal parameters to achieve the required test configuration. The program then sends all the calibrated data to the simulator via the RS232C link. When prompted by the operator, the program signals the simulator to activate the test configuration to perform the test sequence stored in memory and at the time intervals specified by the operator.

As the test progresses the data is automatically logged. In addition the operator can select the data to be output to a disk file or printer.

During test runs the operator has the option of resetting the simulator, restarting the test sequence, setting the simulator to measure either input or output signals and of displaying all parameters most recently sent to the simulator from the control program.

5 USING THE SIMULATOR CONTROL PROGRAM

This section of the document provides an operational description of the simulator control program.

For descriptions of the software modules and subroutines reference should be made to Sections 6 and 7 respectively.

5.1 Program operating procedures

NOTE: *With all operator responses, the program will accept **either** upper or lower case characters.*

5.1.1 Invoking the program

The program name is SIM. To invoke the program from DOS ***enter "sim" at the DOS prompt.*** Once the program is invoked, the following screen message will be displayed.

IONOSPHERIC SIMULATOR CONTROL PROGRAM

Version 1.1

developed by:
P.L. Slingsby
August 1988
Terrestrial Transmission Systems
Electronics Research Laboratory
PO Box 1600
Salisbury
South Australia 5108

Next depress function key *F1 to display an overview* of the program. This overview consists of six screen pages that may be *stepped through by pressing any key to continue.* When the last overview page has been displayed the program will automatically continue. If it is desired *to exit from the program overview before the last page has been displayed, select ESC* to advance to the next stage of program execution.

If an *overview of the program is not required, select any key* (other than F1). The screen will then display the question

Input by keyboard <K> or file <F>?...

Enter *K to create a new set of data* or *F if the input is to be recalled from disk*

NOTE: *Any other key depressed at this point will result in no reaction from the program.*

5.1.2 Entering input from the keyboard

If K was selected when the program was invoked the following prompt will be displayed

one <1> or two <2> paths?...

This prompt allows the operator to simulate either non-diversity or diversity operation.

5.1.2.1 Entering parameters

If *non-diversity (1) is selected* the following screen prompt will be displayed.

```
ENTER PARAMETER VALUES...
Relative level of ground wave 1?(dB,0 is absolute)   {gw1}  __
Relative level of skywave <1>?  (dB,0 is absolute)   {sw1}  __
Relative level of skywave <2>?  (dB,0 is absolute)   {sw2}  __
Relative level of skywave <3>?  (dB,0 is absolute)   {sw3}  __
Relative level of skywave <4>?  (dB,0 is absolute)   {sw4}  __
Frequency offset for path 1?    (0 -> 5000Hz)        {fo1}  __
Signal/noise for path 1?        (0 - 70 dB)          {sn1}  __
Signal/interference for path 1? (as for s/n)         {si1}  __
Delay for skywave <1>? (0 - 11.6ms in 0.1ms steps)   {ds1}  __
Delay for skywave <2>? (0 - 11.6ms in 0.1ms steps)   {ds2}  __
Delay for skywave <3>? (0 - 11.6ms in 0.1ms steps)   {ds3}  __
Delay for skywave <4>? (0 - 11.6ms in 0.1ms steps)   {ds4}  __
Rayleigh fade rate for skywave <1>? (0 - 80 Hz)      {fs1}  __
Rayleigh fade rate for skywave <2>? (0 - 80 Hz)      {fs2}  __
Rayleigh fade rate for skywave <3>? (0 - 80 Hz)      {fs3}  __
Rayleigh fade rate for skywave <4>? (0 - 80 Hz)      {fs4}  __
F1-HELP
```

If *diversity (2) is selected* the following screen prompt will be displayed.

```
ENTER PARAMETER VALUES...
Relative level of ground wave 1?(dB,0 is absolute)   {gw1}  __
Relative level of skywave <1>?  (dB,0 is absolute)   {sw1}  __
Relative level of skywave <2>?  (dB,0 is absolute)   {sw2}  __
Relative level of groundwave <2?(dB,0 is absolute)   {gw2}  __
Relative level of skywave <3>?  (dB,0 is absolute)   {sw3}  __
Relative level of skywave <4>?  (dB,0 is absolute)   {sw4}  __
Frequency offset for path 1?    (0 ->+/-5000Hz)      {fo1}  __
Frequency offset for path 2?    (0 ->+/-5000Hz)      {fo2}  __
Signal/noise for path 1?        (I,0 or in dB)       {sn1}  __
Signal/noise for path 2?        (I,0 or in dB)       {sn2}  __
Signal/interference for path 1? (as for s/n)         {si1}  __
Signal/interference for path 2? (as for s/n)         {si2}  __
Delay for skywave <1>? (0 - 11.6ms in 0.1ms steps)   {ds1}  __
Delay for skywave <2>? (0 - 11.6ms in 0.1ms steps)   {ds2}  __
Delay for skywave <3>? (0 - 11.6ms in 0.1ms steps)   {ds3}  __
Delay for skywave <4>? (0 - 11.6ms in 0.1ms steps)   {ds4}  __
Rayleigh fade rate for skywave <1>? (0 - 80 Hz)      {fs1}  __
Rayleigh fade rate for skywave <2>? (0 - 80 Hz)      {fs2}  __
Rayleigh fade rate for skywave <3>? (0 - 80 Hz)      {fs3}  __
Rayleigh fade rate for skywave <4>? (0 - 80 Hz)      {fs4}  __
F1-HELP
```

The operator must now *enter the desired parameter information at each level of the prompt* as it progresses down the parameter list. As each *value is entered the ENTER key must be selected. Use the backspace to edit keying mistakes.*

As can be seen from the displayed prompts, legal values are given for each parameter. Value "0" indicates that there is no path, therefore if skywave 1, for example, was given the value 0 then assigning a delay to skywave 1 would have no effect. The character "I" may also be used for signal-to-noise and signal-to-interference parameters to indicate a value of infinity.

If an invalid value is entered it is erased automatically and the prompt "re-enter" displayed in its place for two seconds. The operator must then enter the correct value for that parameter. At any stage during the process of entering the parameters the operator may *invoke the HELP option by selecting F1.* An explanation of the valid parameter limits can then be viewed.

NOTE: *If it is required to edit a value already entered and accepted by the program then all subsequent parameter values must be entered before the mistake can be corrected. To edit select function key F3.*

5.1.2.2 Entering the test command

Once the last parameter entry has been made, one of the following prompts will appear at the bottom of the screen. (The first prompt is displayed in the nondiversity case, the second in the diversity case.)

Enter test command: <param> <init val> <final val> <inc><time>

or

Enter test command: <param[&param]> <init val> <final val> <inc> <time>

The operator must *now enter a test command in the requested format.* A valid test command comprises the following:

<param> = the parameter code - a valid test command of the parameter to be changed during the test, as displayed under the "Enter Parameters Values" prompt, eg. sn1 for signal/noise path 1.

<init val> = the initial value that the parameter is to take (in the units specified in the prompt, eg. for sn1, dB).

<final val> = the final value (in specified units) that the parameter is to take.

<time> = the time that each incremented stage of the test is to last. Enter in minutes as a real number, eg. for 5 min 15 s, enter 5.25 min.

In the case of diversity operation the operator may also request that two parameters be changed simultaneously **(the <param[& param]>command).**

This facility allows the operator to test the behaviour of the two paths under different conditions but over the same incremental values of a separate parameter, eg. to test the behaviour of two paths with different delays as the signal-to-noise is stepped from 0dB to 60dB for both paths.

Thus the test command "sn1 & sn2 10 60 5 5", requests that the signal-to-noise ratio of paths 1 and 2 be stepped from 10dB to 60dB in 5dB increments every 5 minutes and is a valid command for the diversity case. It would not be valid for the non-diversity case, however, because sn2 is an invalid command for this mode and also because the "&" character is not allowed in nondiversity command lines.

If an invalid command line is entered then an error message will appear for one second and the operator will be instructed to re-enter the command line. The operator may consult the HELP menu, if required by selecting the function key F1.

5.1.2.3 The Utilities Options

Once the command line entered by the operator has been accepted as valid the following display of the available utilities options will appear on the screen.

F1 - HELP
F2 - SAVE AS TEST <..>
F3 - CHANGE PARAMS
F4 - FILE
F5 - DISPLAY
F6 - REENTER COMMAND
F7 - ADD OSC TONE TO TESTS
F8 - ADD PATH GAIN

A description of each option is given below.

F1 - HELP

This option is available from almost anywhere in the program. It comprises a number of menu and sub-menu options that can be invoked to display information on keyboard responses at various stages throughout the program execution, utility selection and other menu options. The operator can ***exit from the HELP option at any time by selecting ESC .***

F2 - SAVE AS TEST

This option saves all the parameter and command line data currently displayed on the screen as well as the current values for oscillator tone and path gain(s) entered under options F7 and F8 - (these values are 0 and unity respectively by default). When invoked, option F2 causes the screen message **"Saving Test <test number>"** to be displayed. After a test has been saved it can still be changed using the other options described below.

NOTE: *If a test is not saved before proceeding further that data will be overwritten by subsequent data.*

F3 - CHANGE PARAMS

This option enables the operator to edit the previously entered parameter information that is displayed on the screen. The 'up' and 'down' arrow keys are used to move the cursor to the relevant entry where the new data is to be entered. ***When editing is complete select ENTER*** and the program will prompt the operator to reenter the appropriate command line. When this has been entered the program resumes.

F4 - FILE

This option files all the tests that have been entered and saved. When selected the following prompt is displayed.

"Do you wish to change this data? (Press <Y>, <N> or F1 to display tests)"

If it is required to change data, ie. Y is selected, the following prompt is displayed:

"Which test? (Enter number of test, F1 to display all, or ESC to exit)"

Here the operator must enter the digit corresponding to the number of the test to be edited. ***If unsure of the number select F1 to display all saved tests.*** If an invalid number, ie. a test that does not exist, is entered the following error message will be displayed:

"There is no test <*test number*> in this file! Press any key to return..."

Once a valid test number has been entered, the following screen prompt appears.

"Do you wish to change params <P>, oscillator tone <T>, path gain <G>, or command line <C>?.."

This option gives the operator the opportunity to change the information to be filed. ***Select P to edit the parameter information, C to edit the test command line, T to edit the oscillator tone frequency and level, or G to edit the path gain.***

If P is selected the relevant parameter proforma is displayed with the current parameter values shown. The operator then simply edits this list using the 'up' and 'down' arrow keys to move the cursor to the relevant parameter value. ***Select ENTER when editing has been completed.***

If C is selected the command line prompt relevant to the current mode (diversity or nondiversity) is displayed. The operator then simply enters the new test command.

If T is selected the operator is prompted to enter the oscillator frequency and level (in volts).

NOTE: *Once the oscillator tone has been changed, the new data remains valid for all subsequent tests until changed again. To remove the oscillator tone for a subsequent test, the level must be changed to 0 Volts.*

If G is selected the operator is prompted to specify which path gain should be changed from unity (the default), ie. ***for signal select "S", for interference path select "I".***

After the requested changes have been made and accepted by the program, the new data is displayed to the operator in the form shown under F5-DISPLAY below.

The operator is then asked whether additional editing is required. If further editing is required (Y selected) the above procedure is repeated. If no further editing is required (N selected) the following prompt is displayed.

"What file name shall this data be filed under? Press <ENTER> for no save..."

If the operator selects a file that already exists a prompt will appear to ask whether the existing file should be overwritten.

F5 - DISPLAY

This option permits all the data that have so far been saved using option F2 to be displayed. The format is as shown in the example below. Since only two tests are displayed at a time the operator must step through the screen to show all the test data stored.

TEST	1	2
gw1	1.0	20.0
sw1	2.0	20.0
sw2	0.0	10.0
sw3	0.0	0.0
sw4	0.0	0.0
fo1	50.0	50.0
sn1	20.0	25.0
sl1	Infinite	30.0
ds1	0.55	0.25
ds2	0.0	0.5
ds3	0.0	0.0
ds4	0.0	0.0
fs1	10.0	10.0
fs2	0.0	10.0
fs3	0.0	0.0
fs4	0.0	0.0
gw2	0.0	20.0
fo2	0.0	0.0
sn2	0.0	25.0
sl2	0.0	Infinite
osc freq	5.0	0.0
osc level	1.0	0.0
test command	sn1 20 60 5 5	gw1&gw2 5 20 5 5

When viewing is complete select ENTER to continue to the next stage of program execution.

F6 - REENTER COMMAND

This option allows the operator to reenter the command line for the current test. If this option is invoked at a stage of the program execution when it is not available, then it is ignored. When this option is selected the previously entered command line disappears and is lost. ***If this option is invoked by mistake, the command line must be reentered.***

F7 - ADD OSC TONE TO TESTS

This option allows the operator to add an oscillator test tone to the input. The operator is initially prompted for the desired frequency and level (between 0 and 5V). This tone will be digitally added to the input signal. The operator is next prompted to specify the time period (in seconds) over which measurement of the input signal is to be made by the simulator. This measurement is required so that the simulator will be rescaled and recalibrated to allow for the additional tone.

F8 - ADD PATH GAIN

This option allows the operator to alter the gain of the programmable gain unit at the front end of the simulator. By default this parameter is set such that the input gain will be unity, however, it may be adjusted between 1 and 100 (in integer values only). As the requested gain may not equal that calculated by the program (since not all gain values are physically available), the gain achieved is displayed to the operator. Again, measurement of the input signal will be necessary for the purposes of rescaling and recalibration so the operator is prompted to specify the time period over which this measurement should be averaged.

5.1.3 Entering input from a file

If, at the prompt **"Input from keyboard <K> or file <F>?..."**, F was selected, indicating that input should be from a disk file, then the following menu options will be displayed on the screen.

F1 - HELP
F2 - LIST
F3 - VIEW
F4 - EDIT
F5 - SAVE
F6 - DELETE
F7 - ADD TEST
F8 - RENAME
F9 - CONT

A description of each of the above options follows.

F1 -HELP

This option performs the same function as that described above for the utilities options (page 10).

F2 - LIST

This option allows all files in the test directory to be listed, together with their creation date and time. After this option is selected the program returns to the point where the operator is prompted to indicate whether input is to be via file or keyboard.

F3 - VIEW

This option allows the operator to view the data in a particular file. The operator is prompted to enter the name of the file to be viewed. Once the name is entered the contents of the file are displayed on the screen in the same format as the DISPLAY option of the Utilities Options (page 12). After the operator has finished viewing the relevant file the program returns to the initial stage where the operator is prompted to indicate whether input is by file or keyboard. This facility allows the operator to enter new data if it is found that no appropriate file exists.

F4 - EDIT

This option provides the means by which the operator can edit the data stored in a particular file. The operator is first prompted to enter the name of the file to be edited. Next the operator is prompted with questions regarding which test of that file is to be edited, which type of data within that test is to be edited and so on, in the same fashion as the Utilities Options function F4 - FILE described on page 10. It should be noted that changes made to this file are not changed on the hard copy but are stored in memory to be used on this run of the test(s) only. To record these changes permanently it is necessary to invoke the SAVE option F5. The program therefore returns to the file menu for this reason.

F5 - SAVE

When this option is invoked the operator is prompted as to whether any further editing is necessary. If not, then the operator is prompted to supply the name of the file to which the edited data should be saved. As with the keyboard data case, the operator has the option of not saving the changes. If the file named already exists then the operator is queried as to whether the old file should be overwritten.

If there has been no alteration made to the original file data then a message indicating this is displayed on the screen.

F6 - DELETE

Selection of this option deletes the file named by the operator. Before the deletion occurs, however, the operator is prompted to confirm the intention to delete the named file, by the on-screen prompt,

"Delete *filename*?**[Y/N]..."**

If Y is selected the file is deleted. If N or any other character is entered at this stage the command is ignored and no further action is initiated.

F7 - ADD TEST

This option allows the operator to add a test(s) to the end of an existing file. The operator is first prompted to supply the relevant file name. The program then changes to the keyboard entry mode and parameter data may be entered in the same way as described on page 8 for keyboard entry. All the Utility Options described previously are also available to the operator to facilitate data entry. Unlike the keyboard case, however, the program does not proceed to run the tests after all editing has been completed. Instead it returns to the file menu.

F8 - RENAME

This option enables the operator to rename a file. When first selected the operator is prompted to enter the old file name. Next the operator is prompted to enter the new file name. This renames the file.

F9 - CONT

This option provides the only means to continue past the file menu when in the "input via file" mode. Since many files may have been entered into memory through the various editing options, the operator is first prompted to indicate the name of the file containing the tests that are to be run. These tests are then loaded into memory. If an invalid file name is entered by the operator or the named file has, for some reason, not been able to be opened, then an error message is displayed and the operator is prompted to re-enter the relevant file name.

5.1.4 Initiating communication with the simulator

With the data for the test(s) to be run now in memory (having been entered either by the keyboard or from file), the program now needs to determine where the logging data should be sent. The following prompt is now displayed to the operator.

"Test data log to printer <P>, to file <F>, or to screen <default>?..."

NOTE: By default the logging data is always sent to the screen. The operator, however, has the option of selecting either "P" to indicate that the data should also be recorded on a printer (connected to parallel port 1), or F to indicate that the data should also be logged to a file.

If F is selected (the data is to be logged to a file) the operator is next alerted by the following screen prompt

"What file name?".

The name of the file to which the logging information is to be sent should then be entered.

NOTE: The program accepts any file name that is valid under DOS.

After entering the file name the operator is prompted as follows to specify whether the test (or sequence of tests) should be repeated continuously or executed once only.

"Do you want these tests to continue indefinitely [Y/N]??"

The default response is N (no). It should be noted that under these circumstances, although the program will cease executing after the test(s) in memory have been executed once, the simulator will remain operating and set up to the last test configuration that was run.

If the Y (yes) response was selected, ie. the continuous execution mode is invoked, then the following on-screen prompt is displayed.

"<<<The tests may be stopped by pressing F10-Quit>>>".

5.1.5 Simulator input measurement

After the operator has specified where the logging information is to be sent, and has specified a continuous or once only mode of test operation, there is a small delay while the parameter data is transferred to the simulator via the RS232C link. When this data has been successfully communicated the following on-screen prompt appears.

"Apply signal and enter RMS period (secs) for INPUT measurement..."

NOTE: The simulator needs to make internal measurements of the input signal (and the interference signal, if applicable) so that it can calibrate path parameter values. It is therefore important that the signal level measured by the simulator is that which will be applied to the simulator during the test sequence - a measurement of input signal levels must be made each time the level is changed to ensure that the desired test configuration is set up.

The operator must next specify the time period (in seconds) over which this measurement should be made. A measurement is made by the simulator every second so that an operator response of "5" would cause the simulator to make five measurements of the input signal and average the results over five seconds.

Instead of a number, the operator may enter the character "C". This causes the simulator to perform measurements continuously until signalled (by any operator keystroke) to stop. Each measurement made in this way is displayed on the screen, thus allowing the operator to adjust the signal level whilst noting the simulator readings.

Next the simulator measures the interference input level. The operator is again prompted, as follows, to enter the period over which this measurement should be averaged.

"Apply signal and enter the RMS period (secs) for INTERF measurement..."

The same options are now available to the operator as were available for measurement of the input signal. It is important that measurement of these two signals is correct since they form the basis for calibration of the simulator internal parameter values.

NOTE: On occasions the simulator may mis-measure these signals. It is important therefore to ensure that these values look correct, otherwise the test configuration will not be that requested and it will have a different signal-to-noise ratio than expected.

5.1.6 Test set up

Once the input and interference measurements have been entered, the test(s) is now ready to begin. The following on-screen prompt will be displayed.

"Test <*test number*> is now ready to start.
Press any key to begin..."

With the 'begin' function initiated, the test progression is then logged to the screen. The current test parameter values, the code for the parameter(s) being incremented, the current value for that parameter(s) and the time at which the current stage of the test sequence was initiated are all displayed to the operator.

NOTE: The starting time for the test(s) is not the time when the operator signals the controller to start, but is the instant recorded in the time column of the logged data. The variable nature of the simulator asynchronous communication with the controller program makes it impossible to accurately predict the time interval between when the program signals the simulator to start and when the simulator actually responds.

5.1.7 Menu options

As the test(s) progress the following menu options appear at the bottom of the screen. They can be invoked by the operator at any time available to the user.

F1-HELP F2-RESET F3-BOOT F4-DISPLAY F5-MEASURE F6-RESTART F8-GAIN F10-QUIT

A description of each option is given below.

NOTE: If a menu option is invoked at a time when the test is timed to alter a parameter(s), the test will not implement the change until the menu option has been executed. This means that although the operator can invoke menu options whilst tests are in progress without disturbing their automatic progression, the possibility of interrupting a test transition does exist. By observing the test progression the operator can lessen the likelihood of interrupting the test.

F2-RESET

This option allows the operator to manually reset the simulator, thus allowing a re-initialisation of the fading, noise and phase conditions. Perhaps not so relevant in an automated system, this option can nevertheless be useful when testing different equipment with the same sequence of fading and noise when the system is utilised in manual mode. The only evidence the operator is given of action taken by this option is a an audible discontinuity in the audio output of the simulator.

F3-BOOT

Selection of this option sends all the current parameter values for the current test stage to the simulator. This enables the operator to restore the simulator conditions if, for example, the operator suspects that the desired test configuration has not been achieved.

F4-DISPLAY

When this menu option is invoked two further options (F1 and F2) are displayed to the operator. By selecting F1 the operator can view the current test parameter values - these are displayed in the format shown on page 12. By selecting F2 the values of the sixty four parameters last sent to the simulator are displayed.

F5-MEASURE

With this option the operator can instruct the simulator to make internal signal measurements.

When this option is selected four further options (F1, F2, F3 and F4) are displayed to the operator.

- To measure the input signal select option F1
- To measure the interference signal select option F2
- To measure output signal 1 select option F3
- To measure output signal 2 select option F4

Once the relevant signal has been selected the operator is next prompted to enter the period over which the measurement should be averaged. This is performed in the same way as the simulator input measurement selections described in paragraph 5.1.5 above.

NOTE: If continuous measurement is required, again enter C.

Measurements made by the simulator are displayed on the screen in rms volts for the input signals, or in dB for the output signals. The program will automatically signal the simulator to recalibrate its internal path parameters to reflect any change in input level.

F6 - RESTART

This option allows the operator to restart the current test. This is a very useful option when trying to synchronise the system. After a RESTART command the testing sequence proceeds as normal.

F8 - GAIN

This option allows the operator to set the programmable gain unit in the same manner as the Utilities Option F8-ADD PATH GAIN (refer page 13). It must be noted, however, that since disk files do not store the programmable gain value , each time a file is recalled it is assumed to be the default value of unity. The operator has, of course, previously been given the opportunity to set the path gain (whether input was from the keyboard or from a disk file).

F10-QUIT

Select this option to exit the program. When selected the operator is prompted to confirm this intention. If confirmation is not correctly given then the request to exit is ignored and program execution continues as before. If the operator confirms the wish to exit then the on-screen warning **"Exiting!!"** flashes for several seconds and the program terminates. Once the program has ceased to execute, whether by selecting the QUIT command or at the completion of a test, the simulator will remain configured with the last parameters that were sent to it by the controller program until switched off.

6 PROGRAM MODULE DESCRIPTIONS

The system controlling program comprises a number of function-specific modules. Figure 3 overleaf provides a functional breakdown of the overall program whilst the associated Figures 4,5,6, and 7 provide breakdowns of the main functions and show the software modules associated with each. Figure 8, in addition to providing a list of the subroutines in each module, also shows the data transfer between modules. Each of the modules is described briefly in paragraphs 6.2 to 6.10 below.

NOTE: For a description of the associated subroutines reference should be made to paragraph 7 below.

6.1 Timing Considerations

It is a requirement of the test bed Pilot Tone Sounder (PTS) that input to the ionospheric simulator be disconnected for five minutes every sixty minutes so that true noise floor measurements of the system can be made. It is therefore necessary that the simulator controller be time-synchronised with the PTS and also that the controller operate an analog switch to disable the input to the simulator for the required five minute period.

NOTE: While the initial controller software included the control of the analog switch, during subsequent development of the project this function was deemed unnecessary.

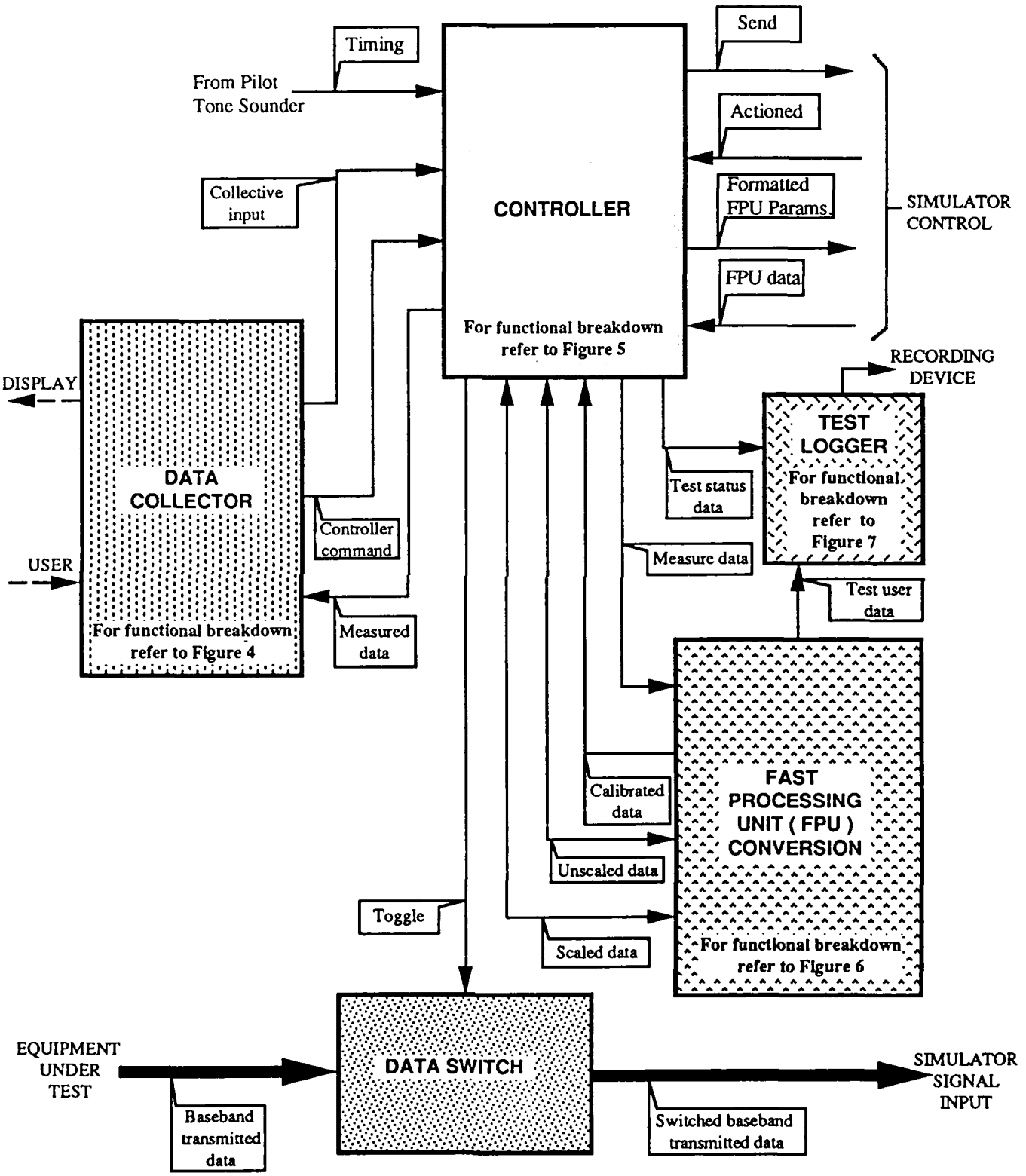

Figure 3 Program functional breakdown

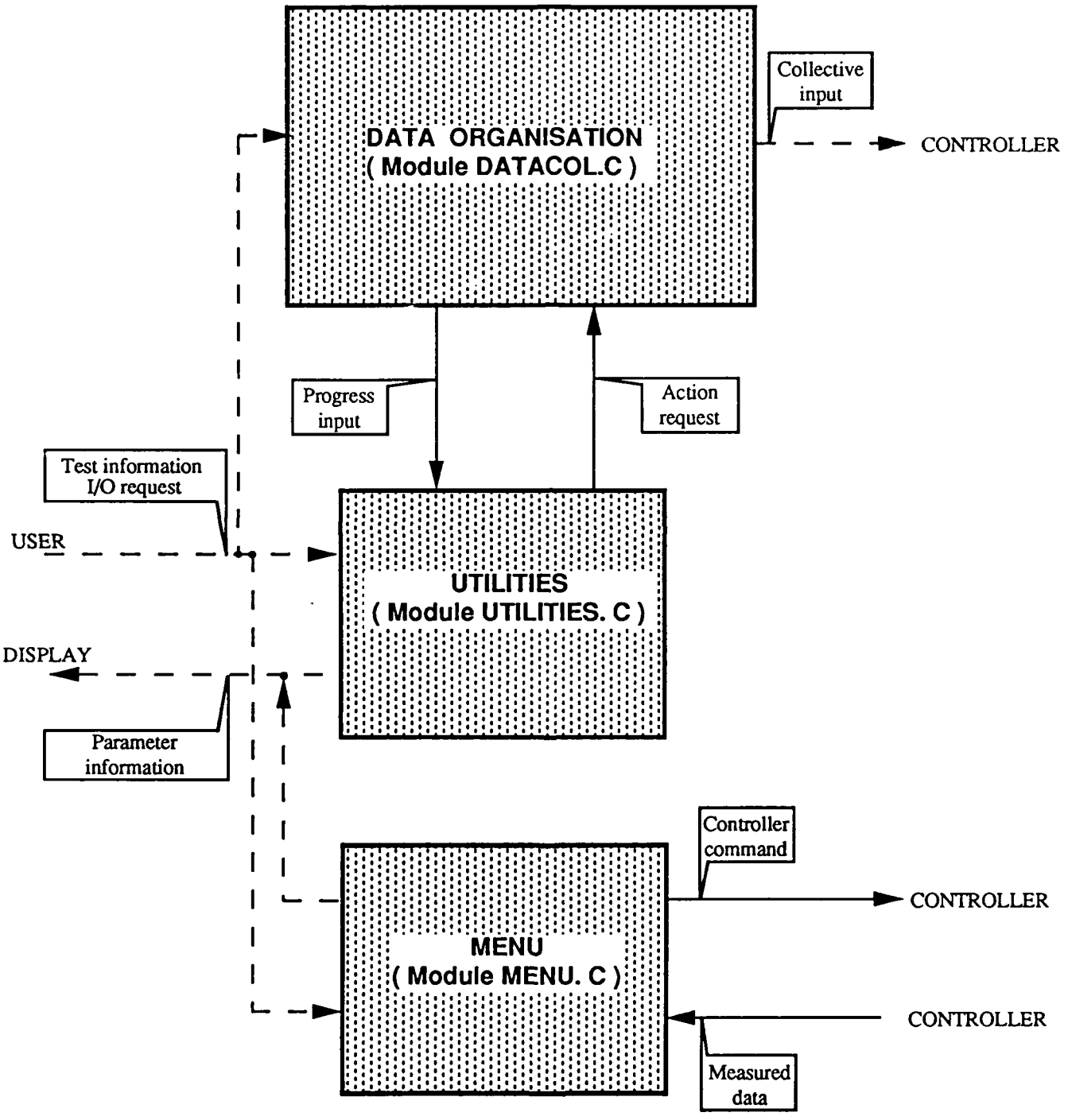

Figure 4 Data Collector functional breakdown

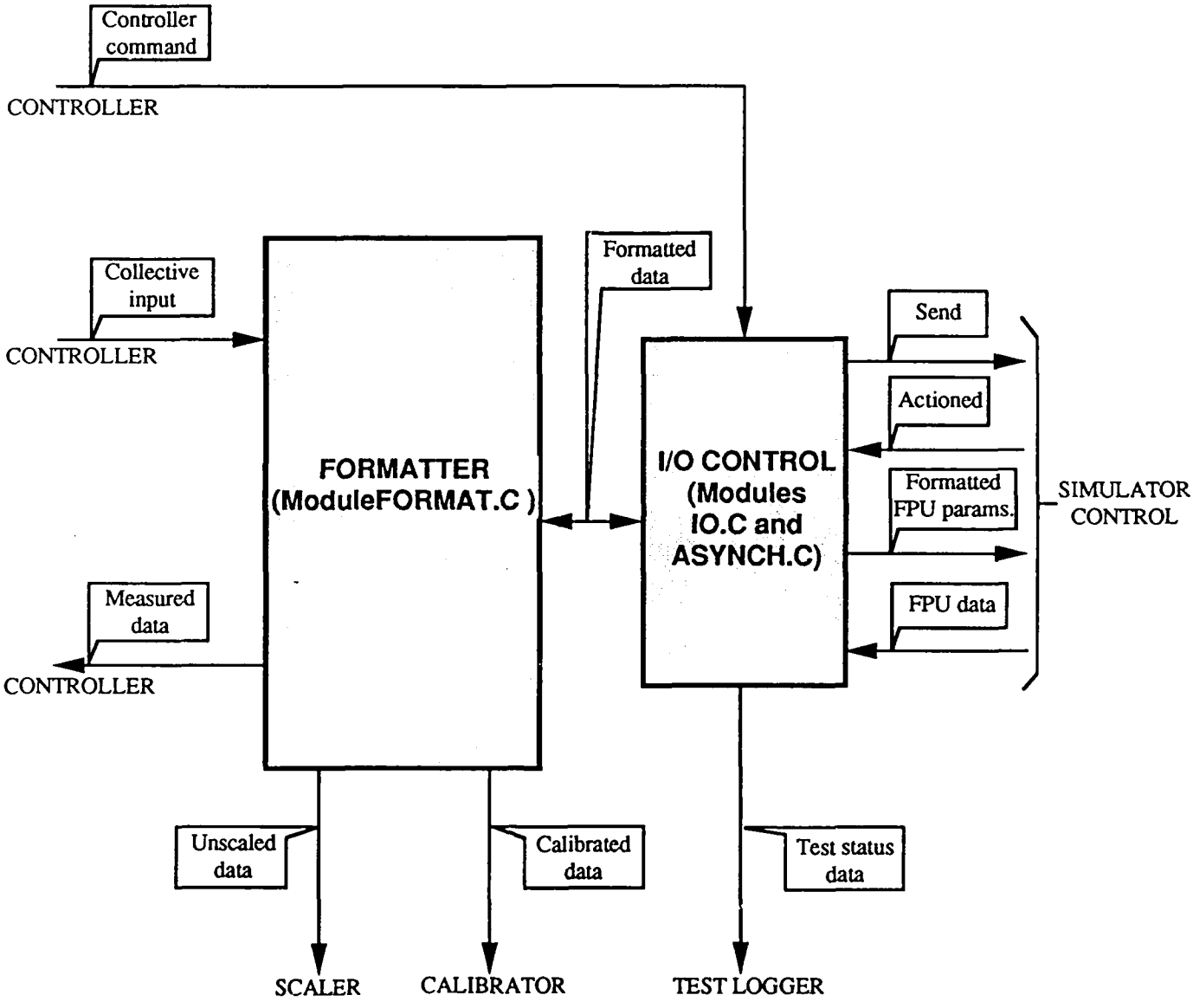

Figure 5 Controller functional breakdown

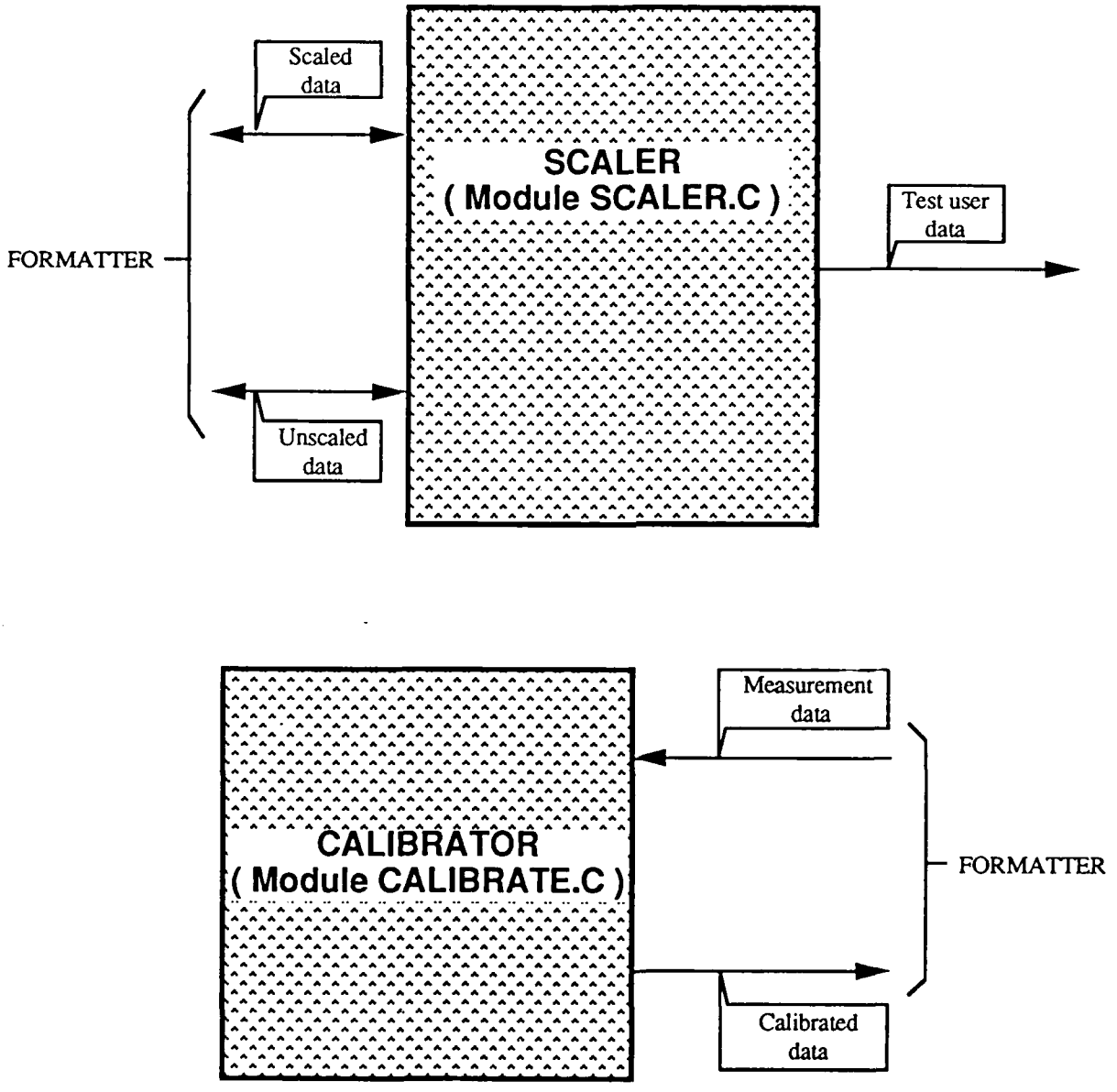

Figure 6 Fast Processing Unit Conversion functional breakdown

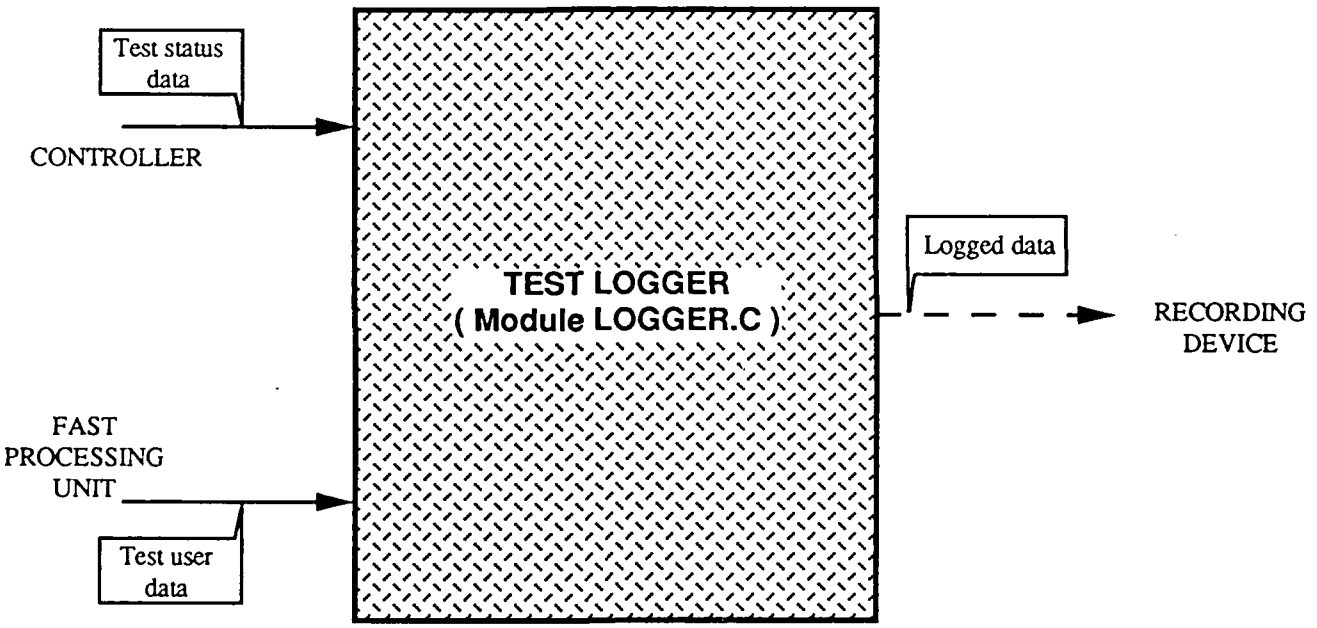

Figure 7 Test Logger functional breakdown

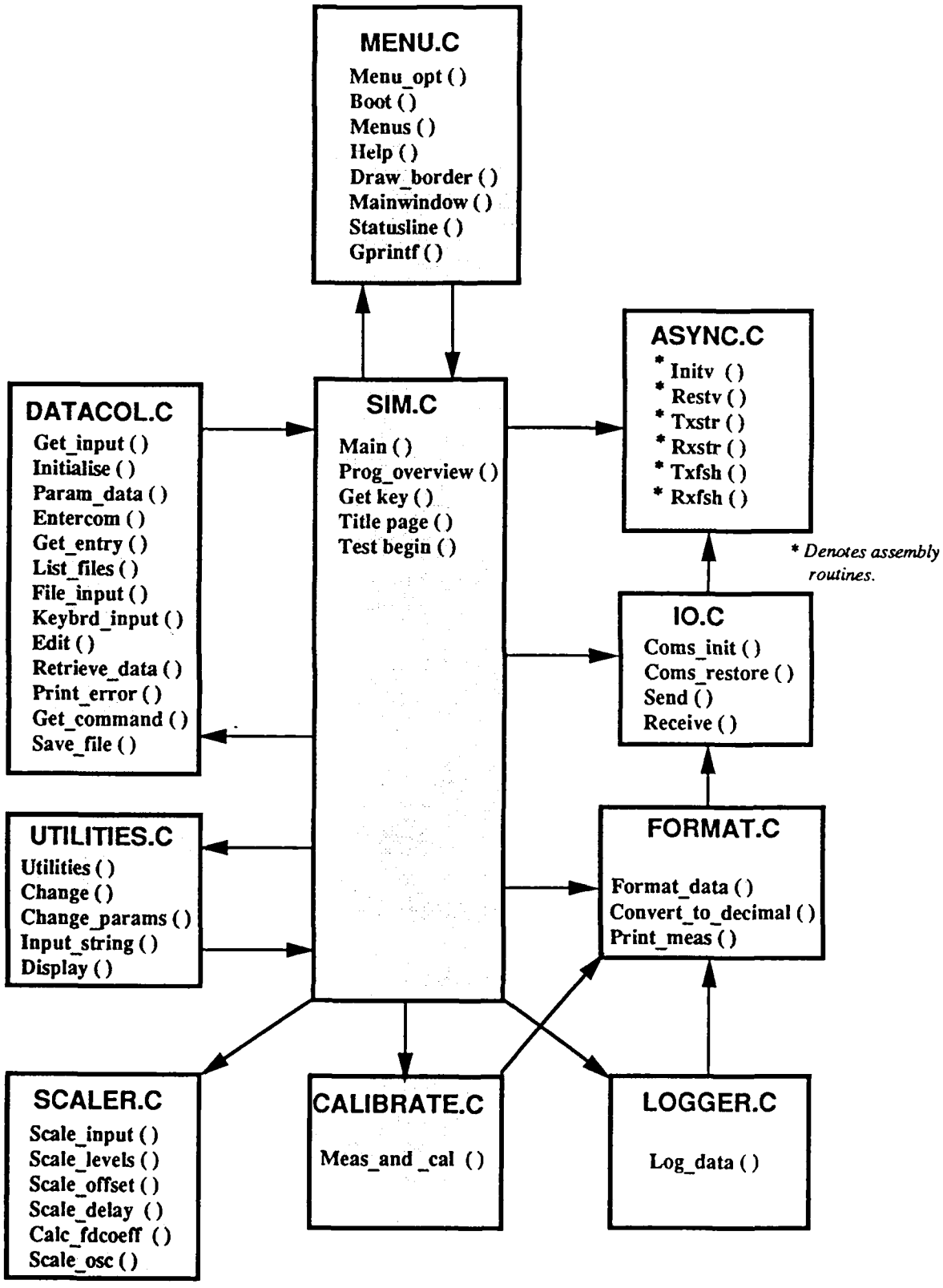

Figure 8 **Module data transfer and subroutine listing**

The controller must also be able to signal to the PTS when the simulator has been reset or when a new test configuration has been implemented. The simulator has a non-negligible response time to commands generated by the controller program and there is also a time delay associated with the transfer of data over the RS232C link. Consequently, there must be a communication link between the controller and the PTS in order to avoid unnecessary corruption of the PTS logged data when the simulator is in a transition state. The impending modifications to the system (referred to in the Introduction), will relieve the need for PTS-controller synchronisation.

6.2 Module SIM.C

This module contains the "Main()" subroutine. It controls the program flow to the DATACOL.C; SCALER.C; IO.C; FORMAT.C; CALIBRATE.C; MENU.C and LOGGER.C modules. It also contains all the routines necessary to set up the graphics windows that are used throughout the program. The test for the overview of the program, which may be invoked at the beginning of a run to display information regarding the use of the program (program overview), is also included in this module.

6.3 Module DATACOL.C

This module contains routines that are used to extract from the operator all necessary information about the automated tests that are to be run. The module is only called from SIM.C, which passes DATACOL.C no data. All input is from the operator, who is given the option of entering the necessary data either from the keyboard or from a disk file. Once all the data has been entered into memory and any necessary editing completed (via module UTILITIES.C) module DATACOL.C passes program control back to "Main()" in SIM.C.

6.4 Module UTILITIES.C

This module contains the routines that allow the operator to edit, display and otherwise manipulate the test data. It is called from DATACOL.C, which passes any necessary parameters for the manipulation that are not already in memory. Depending on the type of manipulation performed, UTILITIES.C may pass an integer back to DATACOL.C.

6.5 Module SCALER.C

This module contains all the routines that scale the values entered by the operator and convert them into the long integers expected by the floating point unit (FPU) of the simulator. It is called only from module SIM.C, which passes to SCALER.C the test parameters that must be scaled and the values that these parameters must take. On completion of the scaling process this module returns to SIM.C.

6.6 Module FORMAT.C

This module contains the routine "format_data()" only. It is called either from module SIM.C or from CALIBRATE.C. The routine constructs the required controller command in an ASCII format acceptable to the simulator. The module is passed an integer representing the instruction type required, and depending on this integer, may be passed additional

data. Once the ASCII instruction has been constructed module IO.C is called to communicate with the simulator.

6.7 Module IO.C

This module contains the routines that control communications over the RS232C link, by invoking the relevant assembly routines of module ASYNC.C. It is called from either Main(), in SIM.C, or from the FORMAT.C module.

6.8 Module ASYNC.C

This module contains the assembly language routines that control the RS232C communications between the simulator and the controller. It comprises routines to initialise communications, to transmit and receive characters, to flush transmit and receive buffers, and to restore the program after communications.

6.9 Module CALIBRATE.C

This module contains the subroutine "meas_and_cal()" only. It instructs the simulator to make a measurement of the simulator input signal(s) and then calibrates the measurement communicated to the controller by the simulator. This value is then sent to the simulator (via module IO.C) so that internal calibrations may be made within the simulator.

6.10 Module MENU.C

This module is called from the SIM.C module to implement user menu requests or to display the help menu, or from the DATACOL.C or UTILITIES S.C. modules. The module is passed information indicating the menu item to be implemented, or the help screen to be displayed. After completion of the necessary action the module does not pass any data back to the calling routine.

Except for module ASYNC.C, each of the above modules is compiled and linked by the Turbo C MAKE file "sim.mak". The "sim.mak" file consists of a number of rules for compiling each of the above modules into object files. It also includes the rule for linking the modules together to form the executable ".EXE" file. By using this utility, only the files that are altered need to be recompiled because Turbo C looks at each object file in a rule and recompiles only if there has been a recent change to that file. Similarly, Turbo C only relinks a program if there has been a change affecting the linking rule(s) included in the ".MAK " file.

Module ASYNC.C is compiled separately from the other modules by the Microsoft assembly compiler MASM 5.1. The object file 'async.obj' is then included in the relinking rule in the file "sim.mak".

NOTE: It is important to remember that every time a change is made to ASYNC.C the file must be recompiled using an assembly compiler, otherwise "sim.mak" will relink the old version.

7 SUBROUTINE DESCRIPTIONS

The following is a brief description of the subroutines contained in each of the modules described in Section 6 above. Should a more detailed description and comprehensive flow charts for each subroutine be required refer to the associated Development Documentation System (DDS) material held by the author.

7.1 Module SIM.C

This module contains the following five sub-routines.

- MAIN ()
- PROG_OVERVIEW ()
- GET_KEY()
- TITLE_PAGE()
- TEST_BEGIN()

7.1.1 MAIN()

Called on program start up. No parameters passed to the function.
Calls functions title_page(), get_input(), coms_init(), send(), scale_input(), format_data(), meas_and_cal(), testbegin(), log_data(), menu_opt(), coms_restore().

Description - Displays title page (function title_page()) then gets input for the test sequence to be run from the operator (function get_input()). Next it initialises programmable gain unit values to unity and installs interrupt service routine and initialises communications over the RS232C link (function coms_init()). Sends a command to initialise the simulator (function send()) and scales all initial parameter values (function scale_input()). Then formats and sends all these scaled parameter values to the simulator (function format_data()). Makes measurement of input signals and calibrates the relevant path data (function meas_and_cal()) then indicates to the operator that the test sequence is ready to be run (function testbegin()). Displays menu options then for each test increment scales, formats and sends to the simulator the relevant parameter data, and then logs the progress of the test sequence (functions scale_input(), format_data() and log_data() respectively). While waiting for test time to elapse checks if a menu option has been chosen by the operator and if so invokes the relevant option (through function menu_opt()).

7.1.2 PROG_OVERVIEW()

Called from function Main(). No parameters passed to the function.
Calls no other function.

Description - Displays up to six screens describing the operation of the program to the operator. The screens may be stepped through or exited from.

7.1.3 GET_KEY()

Called from Main(), utilities(), get_input(), menu().
Calls no other function.

Description - Returns the relevant part of the integer value of the key invoked by the operator.

7.1.4 TITLE_PAGE()

Called from Main(). No parameters passed to the function.
Calls function prog_overview() if instructed by operator.

Description - Displays title screen page outlining program title and authorship. If operator specifies, calls prog_overview(), otherwise returns to main().

7.1.5 TEST_BEGIN()

Called from Main(). No parameters passed to the function.
Calls no other function.

Description - Displays a message to the operator indicating that the test sequence can now be started. Waits for the operator to press a key to return to Main().

7.2 Module DATACOLC

This module contains the following thirteen subroutines.

- GET_INPUT ()
- FILE_INPUT ()
- LIST_FILES ()
- EDIT ()
- RETRIEVE_DATA ()
- KEYBRD_INPUT ()
- INITIALISE ()
- PARAM_DATA ()
- GET_ENTRY ()
- PRINT_ERROR ()
- ENTERCOM ()
- GET_COMMAND ()
- SAVE_FILE ()

7.2.1 GET_INPUT()

Called from Main(). No parameters passed to the function.
Calls functions keybrd_input(), file_input(), getkey().

Description - Operator chooses whether input is from a file, in which case file_input() is called, or from the keyboard, in which case function keybrd_input() is called. If an invalid key is pressed at this point the operator is again prompted to make this choice. The operator then nominates whether the output should go to a file (in which case the file name must be supplied), to a printer, or to the screen (the default). The operator must finally specify whether the test sequence should run continuously.
The program returns an integer signifying the number of tests that were entered into memory (variable "count").

7.2.2 FILE_INPUT()

Called from get_input(). The address of parameter "repeat", indicating whether the current program loop in get_input() should be repeated, is passed to the function. Calls functions getkey(), help() (in module MENU.C), list_files(), retrieve_data(), display() (in module UTILITIES.C), change() (in module UTILITIES.C), edit(), save_file(), initialise(), param_data() and entercom().

Description - Displays file menu options to the operator (refer to paragraph 5.1.3 for description of these options). Sets function loop flag "get_choice" to FALSE and invokes relevant file menu option.

F1-HELP

Saves text, calls help(), then restores text screen and sets "get_choices" to TRUE.

F2-LIST

Calls list_files().

F3-VIEW

Calls retrieve_data() and if this routine returns a non-zero value (indicating that changes were made) calls display().

F4 -EDIT

Saves current text screen, calls retrieve_data() and if this routine returns a non-zero value, calls change(). Then, if change() returns a non-zero value, routine display() is called, followed by edit() and flag "changes" is set TRUE. The saved text is then restored to the screen, the cursor is repositioned and "get_choices" is set TRUE.

F5-SAVE

Calls function change() and if this returns zero, prints an error message and sets "get_choice" TRUE. Otherwise calls save_file().

F6 -DELETE

Prompts operator for name of file to be deleted. If operator confirms intention to delete the named file, constructs file path name and uses Turbo C function 'remove()' to delete the file.

F7 -ADD TEST

Calls retrieve_data() and if the return value from this function is non-zero calls functions initialise(), param_data() and entercom() (placing the return value from this last function in variable "numtests"), and sets "get_choice" to TRUE. If the return value was zero then no action is taken.

F8 -RENAME

Prompts user for old and then new file names, constructs the full path file name of both files and uses the Turbo C function rename() to change the file name.

F9-CONTINUE

Calls retrieve_data() and assigns the return value to variable "numtests".

If flag "get_choice" is now set to TRUE the function loops back to accept another operator file menu option. Otherwise sets main program loop variable "loop" - to FALSE, if the last option invoked was F9 and "numtests" was set to a non-zero value, or else to TRUE.
The function returns the integer variable "numtests".

7.2.3 LIST_FILES

Called from get_input () or from file_input(). No parameters passed to the function. Calls no other function.

Description - Displays all file names in the current test directory (this directory is specified by the macro definition "PATH" included with the module declarations) with creation date and time, by using the Turbo C findfirst() function. The function returns no value.

7.2.4 EDIT()

Called from file_input() or from entercom(). The number of the test to be edited "testnum" is passed to the function.
Calls functions display() and change() (both in module UTILITIES.C) and getkey().

Description - If operator indicates that the tests should be displayed (option F1) then subroutine display() is called, and edit() is recursively called until the operator does not choose to redisplay the tests. If the operator instead chooses to edit the data then function change() is called and if this function returns a non-zero value function display() is called. Function edit() is then recursively called until the operator indicates that no further changes to the data need be made.

7.2.5 RETRIEVE_DATA()

Called from function file_input(). No parameters passed to the function.
Calls no other function.

Description - After the operator has entered the name of the file from which the data is to be retrieved, the full path name is constructed. If the relevant file is not able to be opened an error message is displayed to the operator and the function returns no value to file_input(). Otherwise all relevant file data is read into memory and the parameter "nc" is set accordingly.
The function returns the number of tests read into memory.

7.2.6 KEYBRD_INPUT()

Called from get_input(). No parameters passed to the function.
Calls functions initialise(), param_data() and entercom().

Description - Subroutine initialise() is called to display the relevant parameter proforma and then subroutine param_data() is called to collect all the operator parameter data input. Function entercom() is called to collect the user command line instruction and the return value from this function is assigned to variable "num" which is then returned by keybrd_input () to the calling function.

7.2.7 INITIALISE()

Called from function keybrd_input(), file_input() or from change() (in module UTILITIES.C).
Calls getkey().

Description - If the number of paths for the relevant tests "nc" is zero (indicating that the function was called from keybrd_input()) the operator is prompted to specify the number of paths. The corresponding proforma for the number of paths is then displayed to the operator.
This function returns no value.

7.2.8 PARAM_DATA()

Called from keybrd_input() or from file_input(). No parameters are passed to the function.
Calls functions get_entry() and print_error().

Description - Calls get_entry() for each parameter and places the return value from this function into variable "num". This variable is then checked for legal limits and if outside such limits routine print_error() is called to display an error to the operator and prompt for parameter re-entry. Parameter values for oscillator frequency and level are then initialised. The function returns no value.

7.2.9 GET_ENTRY()

Called by function param_data(). The variable "cursor", containing the current position of the cursor, is passed to the function.
Calls getkey(), help(), print_error() and input_string().

Description - The cursor is positioned at the relevant parameter and if the operator invokes F1 (help) the current text screen is saved, the function help() (in module MENU.C) is called, the text screen is restored to the screen and the cursor is repositioned. If the operator enters an invalid key subroutine print_error() is called and then getkey() is called to get the next user keystroke. Function input_string() is called to enter the operator keystrokes as a string and the macro "value()" (defined in the declarations of module DATACOL.C) is used to convert this string into a floating point value.
The function returns the floating point value of the operator entry.

7.2.10 PRINT_ERROR()

Called from get_entry(), param_data() or from change_params() (in module UTILITIES.C). The cursor position "row" is passed to the function.
Calls no other function.

Description - Positions cursor, displays error message for two seconds and erases error message. No value is returned by this function.

7.2.11 ENTERCOM()

Called from keybrd_input() or from file_input(). A pointer to the variable "count", which contains the total number of tests in memory, is passed to the function.
Calls functions getkey(), get_command(), help(), utilities(), edit() and save_file().

Description - Displays relevant proforma, ie. for either one or two paths, to the operator. Waits for operator keystroke (function getkey()). If F2 is selected this request is ignored. If the operator enters other than a utilities option, then function get_command() is called to collect the command line instruction. If the return value from get_command() is zero, ie. the command was not valid, then entercom() is recursively called; otherwise all utilities options are displayed to the operator

and getkey() is called to input the operator's choice. The function loop flag "recur" is set TRUE if this option is F1 (HELP) or FALSE if this option is F7 (ADD OSC TONE) or F8 (ADD PATH GAIN). For all other options "recur" is set TRUE.

F1-HELP

Text screen is saved, function help() is called and the text screen is restored.

F2-SAVE AS TEST <*test numbers*>

Function utilities() (in module UTILITIES.C) is called and the variable containing the number of tests in memory ("cnt") is incremented. The text screen is then saved and a message indicating that the data is being saved is displayed to the operator for one second. The text screen is then restored and "recur" is set TRUE.

F3-CHANGE PARAMS

Function utilities() (in module UTILITIES.C) is called and then the cursor is restored to its previous position and "recur" is set TRUE.

F4-FILE

Functions edit() and then save_file() are called to allow the operator to make last minute changes and to save the data, respectively. From this option the routine returns.
The value returned is "cnt" - the number of tests currently in memory.

F5-DISPLAY

The current text screen data is saved and if there are no tests currently in memory this is indicated to the operator. Otherwise, function utilities() (in module UTILITIES.C) is called to display all test data. The saved text data is then restored to the screen and "recur" is set TRUE.

F6-REENTER COMMAND

"recur" is simply set to TRUE.

F7-ADD OSC TONE

Function utilities() (in module UTILITIES.C) is called and the cursor is then repositioned to its previous position.

F8-ADD PATH GAIN

Saves the text screen and calls utilities() (in module UTILITIES.C). Then restores the saved text screen and restores the cursor to its previous position.

If flag "recur" has been set then entercom() is called recursively. Otherwise, the cursor is repositioned and the program waits for an operator keystroke. The program then actions the relevant utilities option as above.

7.2.12 GET_COMMAND()

Called from entercom() or from change() (in module UTILITIES.C). Parameters passed to the routine in the function call are; "keystroke" - the integer value of the key last entered by the operator, "i" - a pointer to the array index of the first parameter code entered by the operator, "j" - a pointer to the array index of the second parameter code entered by the operator, "first" - a pointer to the initial (double precision floating point) value that the test parameter(s) will take, "last" - the final (double) value of this parameter(s), "step" - the (double) value of the increment by which this parameter(s) shall be changed, and "time" - the (double) value of the time to be taken for each step increment.

Description - The keystroke made by the operator is first echoed to the screen and the remainder of the command line instruction is read. The test parameter code(s) is checked against all valid codes (held in external array comtable[]). If an "&" character was detected in the command line instruction then variable "j" is set to the relevant code. If an "&" was not detected then "j" is initialised to -1. If an illegal value or parameter code was entered by the operator then an error message is displayed for two seconds and the function returns a value of zero to its calling function. Otherwise the function returns a value of one.

7.2.13 SAVE_FILE()

Called from functions entercom() and file_input(). The variable "numtests" containing the number of tests in memory is passed to this function.
Calls getkey() and input_string().

Description - Loop flag "repeat" is set FALSE. Then the operator is prompted to enter the name of the file to which this data should be sent. Function getkey() is called to return the integer value of the next operator keystroke and if this is RETURN ie. no save is required, then the function returns. Otherwise the file name is read in by function input_string() and the program checks (using Turbo C function findfirst()) that this file does not already exist. If the file does exist then the operator is asked whether that file should be overwritten and function getkey() is called to get the operator response to this prompt. If the operator response was not in the affirmative then "repeat" is set TRUE and the function loops around to the beginning. Otherwise the data is written to the named file. The function returns no value.

7.3 Module UTILITIES.C

This module contains the following five subroutines.

- UTILITIES ()
- CHANGE ()
- CHANGE_PARAMS ()
- INPUT_STRING ()
- DISPLAY ()

7.3.1 UTILITIES()

Called from functions entercom() (in module DATACOL.C), or from menu() (in module MENU.C). A variable number of parameters are passed to this function. The first parameter, an integer representing the function to be performed, is always passed and is stored in variable "func".
Calls functions change_params() and display().

Description - If "func" is F2 or F5 then the next parameter passed in the function call is read into integer variable "count". Descriptions of the subsequent actions taken for the possible values of "func" are as follows.

F2-SAVE AS TEST <*test number*>

Reads successive six parameters passed in the function call (two integers followed by four double precision numbers) and stores these in the relevant elements of the global structure array "test".

F3-CHANGE PARAMS

Calls change_params().

F5-DISPLAY

Calls display().

F7-ADD OSCILLATOR TONE

Prompts operator to enter oscillator frequency and then level. Stores these in elements 21 and 22 of structure array "test".

F8-ADD PATH GAIN

Prompts user to indicate which path (signal or interference) is to have the gain. If an illegal response is entered the function returns, otherwise the operator is prompted to enter the gain required. The actual gain achieved will then be displayed to the operator. This gain is stored in the relevant element of global simulator parameter array u[] and the function returns the value "x" containing an integer representing the path that has a non-unity gain. This is required by the calling function as a measurement of the relevant input signal must next be made to enable the simulator to recalibrate its path parameters.

In all cases (apart from option F8) the function returns no value.

7.3.2 CHANGE ()

Called from utilities() or edit(). The integer value "n", representing the total number of tests in memory, is passed to the function.
Calls functions getkey(), display(), input_string(), utilities(), get_command(), initialise() and change_params().

Description - The operator is prompted to enter the number of the test to be edited and getkey() is called to input the user's response. If ESC is entered then the function returns with a return value of zero, indicating that no changes were made to the data in memory. If option F1 (DISPLAY) was specified then function display() is called to display to the operator all test data currently in memory. The function input_string() is then called to input, as a string, the operator response. If, after conversion to an integer, this value is greater than the total number of tests in memory an error message is displayed and the function returns with a value of zero. Otherwise the operator is prompted to indicate which data is to be edited. The following summarises the actions taken for each legal user response.

T-OSCILLATOR TONE

Calls utilities() then places the entered values for tone frequency and level into the structure "test". The function then returns with a value of one, indicating that changes have been made to the data in memory.

G-PATH GAIN

Calls utilities() then returns with a value of one.

C-COMMAND LINE

Displays relevant command line prompt for the number of paths indicated by the data in memory. Calls getkey() and then get_command() to input the command

instruction entered by the operator. If the command entered was a valid command then all the data for the command instruction is stored in structure array "test" and the function returns with a value of one. Otherwise the function loops around and prompts the operator again to enter a command line instruction.

P-PARAMETER DATA

Calls initialise() (in module DATACOL.C), to display the proforma applicable for the number of paths indicated by the data now in memory and fills it out with that data. Then the function calls change_params() to allow the operator to edit this data and finally the new edited data is stored in place of the old data. The function returns with a value of one.

7.3.3 CHANGE_PARAMS()

Called from change(). No parameters are passed to the function.
Calls functions getkey(), help(), input_string() and print_error().

Description - Calls getkey() to return the next operator keystroke and if this is ENTER returns to change() with a value of zero. The actions taken for other valid operator responses are as follows.

F1-HELP

Saves current text screen, calls help(), restores the text screen and repositions the cursor.

DOWNKEY (Down arrow key)

If not already at the page bottom, moves cursor down a row.

UPKEY (Up arrow key)

If not already at the page top, moves cursor up a row.

DEFAULT

Calls input_string() to read operator keystroke as a string and if "I" was entered (indicating infinite value) assigns a value of 1.0E11. Checks for legality of entry and if invalid calls print_error() to display error message. Otherwise the value is echoed to the screen and the function returns with no value.

7.3.4 INPUT_STRING()

Called from utilities() or from get_entry() (in module DATACOL.C). An integer "k" is passed to the function representing the integer value of the operator keystroke,

and the address of the array str[] in which the entry string will be stored is also passed.

Description - Calls getkey() to input the rest of the entry string, character by character, into array str[]. If an invalid keystroke is made it is ignored.

7.3.5 DISPLAY()

Called from functions utilities() or from edit() (in module DATACOL.C). The function is passed an integer "c" which contains the total number of tests currently in memory.
Calls no other function.

Description - Displays tests, two per page, until the last test has been displayed or until the user enters a key other than ENTER.

7.4 Module MENU.C

This module contains the following eight subroutines.

- MENU_OPT ()
- BOOT ()
- MENUS
- HELP ()
- DRAW_BORDER ()
- MAINWINDOW ()
- STATUSLINE ()
- GPRINT ()

7.4.1 MENU_OPT()

Called from Main(). The function is passed the integer value representing the option that is to be implemented.
Calls functions help(), format_data(), boot(), menus() and meas_and_cal().

Description - Takes one of the following seven possible paths depending on the value of the integer passed in the function call.

F1-HELP

Calls function help ()

F2-RESET

Calls function format_data () in module FORMAT.C.

F3-BOOT

Calls function boot ().

F4-DISPLAY

Calls function menus ().

F5-MEASURE

Calls function menus ().

F8-GAIN

Calls function utilities () and assigns a return value to variable 'point', indicating the path to which the gain should be applied. Then calls meas_and_cal () with function call parameter 'point' to make a measurement of the relevant input.

F10-QUIT

Requests user confirmation and, if this is given, exits the program. Otherwise no action is taken.

The function returns no value.

7.4.2 BOOT

Called from menu_opt (). No parameters are passed to the function.

Calls function send () in module IO.C, and function format_data () in module FORMAT.C.

Description - Displays a message that parameter data is being sent to the simulator and calls function send () to reset the simulator. There is then a delay of three seconds (the reason for this is discussed under section 9) and format_data() is called to send all current parameter values to the simulator.

The function returns no value.

7.4.3 MENUS()

Called from menu_opt (). An integer value represented by the predefined integer 'type' is passed to the function.

Calls functions getkey (), display (), menus () and meas_and_cal ().

Description - A menu is displayed to the operator depending on the value of 'type'.

DISP

The operator can choose from F1 (to display user values) or F2 (to display all scaled parameter values last sent to the simulator). Getkey() is then called to input the operator's choice.

ESC

The function returns.

F1

Calls function display() then calls menus() recursively.

F2

Displays parameters last sent then calls getkey() and if user keystroke is ESC then returns. Otherwise menus() is called recursively.

7.4.4 HELP()

Called from functions get_input(), file_input(), get_entry(), entercom(), change(), change_params(), menu_opt() and menus(). No parameters are passed to the function.
Calls no other function.

Description - Displays menu options to the operator and, depending on operator choice, displays either the relevant text information screen or the relevant sub-menu.
The function returns no value.

7.4.5 DRAWBORDER()

Called from function help().
Calls no other function.

Description - Draws a two-line border around a flood-filled screen.

7.4.6 MAINWINDOW()

Called from function help(). The character string "header" containing the line to appear at the top of the window is passed to mainwindow() in the function call.
Calls function draw_border().

Description - Prints the header string at the top of the screen and calls function draw_border() to set up the window.

7.4.7 STATUSLINE()

Called from function help(). The character string "msg" is passed to statusline() in the function call.
Calls no other function.

Description - Prints the message string contained in "msg" at the bottom of the screen page.

7.4.8 GPRINTF()

Called from function help(). A variable number of parameters are passed in the function call. The first two of these parameters are the integers "xloc" and "yloc", respectively denoting the x-and y-coordinates of the starting position on screen for the character string to be displayed. The character string to be output is passed to the routine at the address denoted by the third function call parameter, "format".
Calls no other function.

Description - This function is used in the same manner as the textual "printf" C statement to print a string to a graphics screen. Cursor row position is then incremented by a standard amount.

7.5 Module FORMAT.C

This module contains three subroutines.

- FORMAT_DATA ()
- CONVERT_TO_DECIMAL ()
- PRINT_MEAS ()

7.5.1 FORMAT_DATA()

Called from main() or from meas_and_cal() (in module CALIBRATE.C). A variable number of parameters are passed to the function. The first, "com", indicates the relevant command option that is to be performed, and the second, "indx", indicates whether all or a particular parameter is to be formatted for transmission to the simulator.
Calls functions send() and receive() (in module IO.C), print_meas() and convert_to_decimal().

Description - The routine takes one of three possible paths of execution depending on the value of "com":

SET

Parameter values are converted into an ASCII string:-

"**P** (*indx value*) **\r\n**"

where *indx* and *value* are two and four characters in length respectively and "\r\n" represents a carriage return followed by a line feed.
These are then sent as arguments to the function send() and simulator response is waited on by a call to function receive(). This option automatically invokes the next option.

RESET

This sends a sequence of four command instructions to the simulator by calling send() and then receive() (in module IO.C). Value "x" is then set to 0.

MEASURE

Calls send() to send a command to the simulator instructing it to initiate the measurement procedure. The measured values are then read and converted from a string of three sets of integers to a double precision floating point number by function convert_to_decimal(). The function then checks that these measurements are valid. If not, the function loops around to read the next measurement data sent by the simulator and received by the controller. If the value received by the controller ("number") is zero, then the peak value is also set to zero; otherwise the peak value is multiplied by 2^{-13}. The measurement voltage is then calculated as:

$$2.5\,(\text{number})^{\frac{1}{2}}/204800$$

If the measurement point is at the output, this value is further multiplied by 5.1641272. If "indx" is zero, ie. measurement is continuous, function print_meas() is called to print out the voltage and peak voltage. Otherwise the measured values will be averaged over the required time period and then displayed. The value of "x" is then set to (1.25 / this voltage). Function send() is again called to instruct the simulator to stop measurement and function receive() is called to receive the simulator response indicating that measurement has ceased.
The function returns the value "x".

7.5.2 CONVERT_TO_DECIMAL()

Called from function format_data(). A hexadecimal number is passed to the function in the character array "hex_num".
Calls no other function.

Description - Converts hexadecimal characters passed to the function in "hex_num" to a long integer. The function then returns that integer to function format_data(). If any of the original hexadecimal characters were null (that is, an

invalid quanitity was measured by the simulator), the function immediately returns the value -1.

7.5.3 PRINT_MEAS()

Called from function format_data(). The function is passed an integer denoting the point in circuit at which measurement was made by the simulator, the measured voltage level, and the peak value of that measurement.
Calls no other function.

Description - Calculates and prints out the relevant measurement from the information given in the function call arguments. If the point for measurement was at one of the inputs then the voltage rms and peak are displayed. If the measurement was at one of the outputs then the rms value is printed and the signal-to-noise ratio is calculated and displayed in dBs.

7.6 Module IO.C

This module contains the following four subroutines.

- COMS_INIT ()
- COMS_RESTORE ()
- SEND ()
- RECEIVE ()

7.6.1 COMS_INIT()

Called from Main(). No parameters are passed to the function.
Calls assembly routines rxfsh(), txfsh(), initv().

Description - Sets the link communication parameters, via Turbo C function bioscom(), then calls each of the routines listed above to flush receive and transmit buffers and install interrupt service routine, respectively.

7.6.2 COMS_RESTORE()

Called from Main() at exit, by the previously installed Turbo C function atexit(). No parameters are passed to the function.
Calls rxfsh(), txfsh() and restv() (in module ASYNC.C).

Description - Calls each of the above routines to flush receive and transmit buffers, and install/restore the original interrupt vector contents.

7.6.3 SEND()

Called from Main(), boot() and format_data(). The address of the pointer variable "str", that contains the data that is to be sent by the function to the simulator, is passed to send().
Calls assembly routine txstr() (in module ASYNC.C).

Description - Until a null character is read, the contents of "str[]" are sent, character by character, to the simulator via function txstr().

7.6.4 RECEIVE()

Called from Main(), boot() and format_data(). The integer value of the response character expected to be read is passed in the function call to receive().
Calls function rxstr() (in module ASYNC.C).

Description - The function calls rxstr() and waits until the response character expected is received. Then, depending on this response character, the function either waits to receive a carriage return/ line feed pair, or reads a character string into array variable "string[]".
The function returns no value.

7.7 Module SCALER.C

This module contains the following six subroutines

- SCALE_INPUT ()
- SCALE_LEVELS ()
- SCALE_OFFSET ()
- SCALE_DELAY ()
- CALC_FDCOEFF ()
- SCALE_OSC ()

7.7.1 SCALE_INPUT()

Called from Main(). Integer variable "p", representing the parameter(s) to be scaled, and double precision floating point variable "val", representing the unscaled value of the relevant parameter, are passed to the function in the function call.
Calls functions scale_levels(), scale_offset(), scale_delay(), calc_fdcoeff() and calc_osc().

Description - Depending on the value of "p" one or all of the above functions are called. In each function call (except for scale_osc() which has no function call parameters) the parameters "p" and "val" are passed and these are represented by "indx" and "value" in each of the called functions.
The function returns no value.

7.7.2 SCALE_LEVELS()

Called from scale_input().
Calls no other function.

Description - The FPU gains are calculated as follows and placed in the relevant elements of global array "u[]":
Nondiversity case:

scale1 = 1 + 1/sn1 + 1/si1
temp1 = gw1 + sw1 + sw2 + sw3 + sw4
u[52] (gain sw3) = 20305 (sw3 / (2xtemp1xscale1))$^{\frac{1}{2}}$
u[53] (gain sw4) = 20305 (sw4 / (2xtemp1xscale1))$^{\frac{1}{2}}$
u[54] (gain gw1) = 20305 (gw1 / (temp1xscale1))$^{\frac{1}{2}}$
u[56] (gain sn1) = 20305 (5 / (sn1xscale1xFB1))$^{\frac{1}{2}}$
u[58] (gain si1) = 20305 (1 / (si1xscale1))$^{\frac{1}{2}}$
u[60] = -1

Diversity case:

scale1 = 1 + 1/sn1 + 1/si1
scale2 = 1 + 1/sn2 + 1/si2
temp1 = gw1 + sw1 + sw2
temp2 = gw2 + sw3 + sw4
u[52] (gain sw3) = 20305 (sw3 / (2xtemp2xscale2))$^{\frac{1}{2}}$
u[53] (gain sw4) = 20305 (sw4 / (2xtemp2xscale2))$^{\frac{1}{2}}$
u[54] (gain gw1) = 20305 (gw1 / (temp1xscale1))$^{\frac{1}{2}}$
u[55] (gain gw2) = 20305 (gw2 / (temp2xscale2))$^{\frac{1}{2}}$
u[56] (gain sn1) = 20305 (5 / (sn1xscale2xFB1))$^{\frac{1}{2}}$
u[57] (gain sn2) = 20305 (5 / (sn2xscale2xFB2))$^{\frac{1}{2}}$
u[58] (gain si1) = 20305 (1 / (si1xscale1))$^{\frac{1}{2}}$
u[59] (gain si2) = 20305 (1 / (si2xscale2))$^{\frac{1}{2}}$
u[60] = 0

u[50] (gain sw1) = 20305 (sw1 / (2xtemp1xscale1))$^{\frac{1}{2}}$
u[51] (gain sw2) = 20305 (sw2 / (2xtemp1xscale1))$^{\frac{1}{2}}$

NOTE: *For explanation of FB1 and FB2 refer to the Cossor CSP1250 technical manual.*

The function returns no value.

7.7.3 SCALE_OFFSET()

Called from scale_input().
Calls no other function.

Description - The frequency offsets for path 1 (fo1) and path 2 (fo2) are scaled and placed in the relevant elements of the global FPU parameter array as follows:

u[10] = fr(fo1) x 2^{31} / 10000
u[11] = in(fo1) x 2^{16} / 10000
u[12] = fr(fo2) x 2^{31} / 10000
u[13] = in(fo2) x 2^{16} / 10000
(Where fr() and in() indicate fractional and integer portions respectively)

The function returns no value.

7.7.4 SCALE_DELAY()

Called from function scale_input().
Calls no other function.

Description - The user delay values are scaled by multiplying them by 10 and rounding them to an integer value.
The function returns no value.

7.7.5 CALC_FDCOEFF()

Called from scale_input().
Calls no other function.

Description - The algorithm for calculating fade coefficients (the Hilbert filter coefficients for the simulator) from the fade rate f are as follows:

(i) $e = f \times 10^{(e-1)}$ (The fade rate is scaled so that it is always in the range 7-100)

(ii) e1 = e / 2000

(iii) $BM = [(1000 / e)^{\frac{1}{2}} \times 0.9574]^{\frac{1}{2}}$

(iv) qa = 2Πxe1x1.32002
qb = 2Πxe1x0.208123
qc = 2Πxe1x0.8742562
qd = 2Πxe1x1.41585
(These are temporary working variables)

(v) c1 = -2 x exp(-qa)xcos(qb)
_c1 = (c1 + 1) x 2^{15}

(vi) u[15+7xi] =(round)_c1 (i indicates which skywave path)
(This element holds the second of the filter coefficients)

(vii) $c2 = [c1/(2\times\cos(qb))]^2$
_c2 = c2 x 2^{15}

(viii) u[16+7xi] = (round)_c2
(This element holds the third of the filter coefficients)

(ix) $c0 = 1 + c1 + c2$
$\_c0 = c0 \times 2^{15} \times BM \times 4$

(x) u[14+7xi] = (round)_c0
(This element holds the first filter coefficient)

(xi) $d1 = -2 \times \exp(-qc) \times \cos(qd)$
$\_d1 = (d1 + 1) \times 2^{15}$
u[18+7xi] = (round)_d1
(This element holds the fifth of the filter coefficients)

(xii) $d2 = [d1/(2x\cos(qd))]^2$
$\_d2 = d2 \times 2^{15}$
u[19+7xi] = (round)_d2
(This element holds the sixth of the filter coefficients)

(xiii) $d0 = 1 + d1 + d2$
$\_d0 = d0 \times 2^{15} \times BM \times 2$
u[17+7xi] = (round)_d0
(This element holds the fourth of the filter coefficients)

The function returns no value.

7.7.6 SCALE_OSC()

Called from function scale_input().
Calls no other function.

Description - The oscillator frequency entered by the operator is scaled by (5000Hz = 32768) and the voltage level is scaled by (1V = 2317).
The function returns no value.

7.8 Module CALIBRATE.C

This module contains the MEAS_AND_CAL() subroutine only.

7.8.1 MEAS_AND_CAL()

Called from Main() and from menus() and menu_opt() (in module MENU.C). The integer variable "indx" is passed to the function indicating the point in circuit at which measurement should be made by the simulator.
Calls format_data() (in module FORMAT.C).

Description - "indx" is multiplied by 2560 and placed in element [63] of global array "u[]". Format_data() is then called to send this value to the simulator. The

operator is then prompted to apply the relevant signal and to specify the measurement period. Format_data() is again called, this time to initiate the measurement. The return value from this function is placed in the double precision variable "gain" which is divided into high and low bytes and placed in the relevant elements of "u[]".
The function returns no value.

7.9 Module LOGGER.C

This module contains the LOG_DATA () subroutine only.

7.9.1 LOG_DATA()

Called from main(). The function is passed no parameters in the function call.
Calls no other function.

Description - The first time the function is called it displays the logging headings at the top of the screen (and printer page or file if applicable). Each subsequent time it is called, the test parameter(s), its incremental value and the time of the test increment are displayed to the screen, (and to the printer/file if specified). To the right of the screen all parameter values for the current test configuration are displayed.
The function returns no value.

8 FUTURE EDITING OF THE SOFTWARE

When making changes to the controller package, it is important to remember that changes involving functions which rely on the inclusion of library or include files, or which in some other way alter the manner by which the routine should be compiled or linked, must be reflected in the sim.mak MAKE file. As an aid to the programmer the include files needed for each routine, are listed alongside the code printouts in the DDS documentation held by the author. These can be made available on request.

Currently the program is set up to allow a maximum number of twenty tests per file. This is an arbitrary limit only and it can be changed by correcting the value of the global variable MAXTESTS in module SIM.C.

To incorporate routines to control an RS232C link over which error counts can be communicated from the error counters to the controller, the assembly routines of module ASYNC.C may be copied (with modified port and register addresses) and routines similar to those of IO.C may be used to interface the assembly code with the Turbo C routines. Since i/o with the error counters should be straight-forward there is no need to use these assembly routines. The error counter-controller communication can instead be written in Turbo C. The simulator-controller link code is written in assembler only for speed problem reasons.

9 COSSOR SIMULATOR PROBLEMS

The following is a brief note on some problems observed with the Cossor ionospheric simulator.

When the simulator is started from cold there appears to be a warming-up period during which the simulator's response to controller commands is sluggish. Under this condition the simulator will sometimes lock up the RS232C link making communication impossible. For this reason a delay has been introduced in the program in places where commands sent to the simulator in succession do not require acknowledgement . Since there is no handshaking between simulator and controller (the link consists of only transmit, receive and signal ground lines) there is no way of telling when the simulator is ready to receive data. While most controller commands initiate an ASCII response from the simulator, some commands do not. If these commands are sent in succession they may cause simulator problems, even if sent several times.

After prolonged communication between the controller and the simulator, the simulator will sometimes jump to an irregular mode. This mode cannot necessarily be recognised from the simulator audible output. It can only be distinguished by an analysis of the data error rates. To date this mode has not been explained. The only way to correct it is to power the simulator off and on several times.

It should also be noted that the response time of the simulator to the controller commands is not constant; response times between <0.1s and >1.5s per command instruction have been observed. Synchronisation with the PTS is therefore difficult when response time is critical.

DISTRIBUTION LIST

	Copy No
EXTERNAL	
Defence Science and Technology Organisation	
Chief Defence Scientist)	1
First Assistant Secretary, Science Policy)	
Defence Science Representative London	Cont. Sht. Only
Counsellor, Defence Science, Washington	Cont. Sht. Only
Scientific Adviser - Defence Central	Cont. Sht. Only
Scientific Adviser - Navy	Cont. Sht. Only
Scientific Adviser - Air force	Cont. Sht. Only
Scientific Adviser - Army	Cont. Sht. Only
Superintendent, Communications Engineering Division	
Engineering Design Establishment	2
INTERNAL	
Electronics Research Laboratory	
Director	3
Chief, Communications Division	4
Research Leader, Communications	5
Head, Terrestrial Transmission Systems	6
Attention: Mr J. Tilbrook	7
Head, Radiowave Propagation	8
Attention: Ms P.L. Slingsby	9-10
Technical Documentation Section, Communications Division	11-12
Libraries and Information Services	
Librarian, Technical Reports Centre, Defence Library Campbell Park (Central)	13
Documentation Exchange Centre Defence Information Services Branch (and Science Liaison)	15
Joint Intelligence Organisation (DSTI)	15
Defence Science and Technology Organisation Salisbury, Main Library	16-17
Librarian Defence Signals Directorate Melbourne	18
Library ARL	19
Library MRL	20
National Library of Australia	21
Australian Defence Force Academy Library	22
United States Defense Technical Information Centre	23-34
United Kingdom Defence Research Information Centre	35-36
Director Scientific Information Services, Canada	37
New Zealand Ministry of Defence	38
Spares	
Defence Science and Technology Organisation Salisbury, Main Library	39-45

these copies are sent via DSC. Exchange so should be indented under their title

CIP